МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

Курсова робота

з дисципліни «Компоненти програмної інженерії»
тема “Файлова система в пам'яті ”

Виконав
Студент III курсу
групи КП-01
Беліцький Олександр Сергійович

Оцінка

_____
(дата, підпис)

Київ 2023

Мета роботи

Опанувати навички програмування мовою Python, опанувати роботу з інструментами для тестування Pytest та Robot Framework. Набути знань та навичок роботи з Docker.

Інструменти виконання роботи

Python - 3.8.10

PyTest - 7.2.0

Flask - 2.2.2

Robot - 6.0.1

Docker - 20.10.12

Linux/Ubuntu - 20.04

Visual Studio Code - 1.74

PyCharm - 17.0.5

Завдання

Завдання 1

Створити in-memory File System (FS). Файлова система складається з 4 типів вузлів:

1. Directory - може містити інші каталоги та файли. Каталог може бути порожнім або містити декілька елементів. Кількість елементів у каталозі має бути <= DIR_MAX_ELEMS

Дозволені операції::

- Create directory
- Delete directory
- List files and subdirectories
- Move file or subdirectory to another location

2. Binary file - незмінний файл, який містить певну інформацію.

Дозволені операції::

- Create file
- Delete file
- Move file
- Readfile (returns file content)

3. Log text file - текстовий файл, який можна змінювати, додаючи рядки в кінець файлу.

Дозволені операції::

- Create file
- Delete file
- Move file
- Readfile (returns file content)
- Append a line to the end of the file

4. Buffer file - це особливий тип файлу, який працює як черга. Деякі потоки надсилають елементи до файлу, інші витягують елементи з файлу. Номер елемента у файлі дорівнює <= MAX_BUF_FILE_SIZE

Дозволені операції:

- Create file
- Delete file
- Move file
- Push element
- Consume element

Завданння 2

З точки зору поточного завдання вам потрібно розробити додаток HTTP (Restful), який дозволить вам виконувати абсолютно ті ж команди, що і в попередньому завданні, використовуючи протокол HTTP.

Слід визначити наступні ресурси:

- /directory
- /binaryfile

- /logtextfile
- /bufferfile

Завдання 3

Додаток з Завдання 2 запустити в докер контейнері.

Створити додаток - клієнт. CLI - command line interface.

Код виконання завдань

Завдання 1

BinaryFile.py

```
from directory import Directory
from logtextfile import LogTextFile

class BinaryFile:
    #Constr
    def __init__(self, directory, logg):
        self.__name = "BinaryFile.bin"
        self.context = "Something is here"
        self.__directory = directory
        directory.list.append(self)
        self.log = logg
        self.log.append_context("\n" + self.get_name() + ": created")

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name + ".bin"
        self.log.append_context("\n" + self.get_name() + ": was renamed")

    #Move
    def move(self, new_repo):
        new_repo.list.append(self)
        self.__directory.list.remove(self)
        self.__directory = new_repo
        self.log.append_context("\n" + self.get_name() + ": moved to " +
new_repo.get_name())


    #Read
    def get_context(self):
        return self.context

    def get_direcrory_name(self):
        return self.__directory.get_name()

    # Destruc
    def delete(self):
        if(self.__name == "None"):
            raise FileExistsError("Binary file not exists")
```

```
        self.__directory.list.remove(self)
        self.append_context("\n" + self.get_name() + ": was removed")
        self.__name = "None"
```

BufferFile.py

```
from directory import Directory
from logtextfile import LogTextFile

class BufferFile:
    #Constr
    def __init__(self, size, directory, log):
        self.__name = "Buffer.buf"
        self.__size = size
        self.list = list()
        self.__directory = directory
        directory.list.append(self)
        self.log = log
        self.log.append_context("\n" + self.get_name() + ": created")

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name + ".buf"
        self.log.append_context("\n" + self.get_name() + ": was renamed")


    #Move
    def move(self, new_repo):
        new_repo.list.append(self)
        self.__directory.list.remove(self)
        self.__directory = new_repo
        self.log.append_context("\n" + self.get_name() + ": moved to " +
new_repo.get_name())



    #Read       TODO
    def get_context(self):
        return self.list

    def get_direcrory_name(self):
        return self.__directory.get_name()


    #Append
    def append_queue(self, item):
        if len(self.list) == self.__size:
            raise OverflowError("max size reached")
        self.list.append(item)
        self.log.append_context("\n" + self.get_name() + ": append queue")


    def first_out(self):
        self.log.append_context("\n" + self.get_name() + ": poped")
        return self.list.pop(0)



    # Destruc
    def delete(self):
        if(self.__name == "None"):
```

```
            raise FileExistsError("Buffer not exist")
        self.__directory.list.remove(self)
        self.append_context("\n" + self.get_name() + ": was removed")
        self.__name = "None"
```

Directory.py

```
class Directory:

    #Constr
    def __init__(self, name = "autodir"):
        self.__name = name
        self.list = list()

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name

    #Move
    def move_repository(self, new_repo):
        if new_repo in self.list:
            self.list.remove(new_repo)
            new_repo.list.append(self)
            return
        new_repo.list.append(self)
        self.directory = new_repo

    #Read

    def sort_list(self):
        new_dir_list = []
        for item in self.list:
            if not item.get_name().endswith(".bin") or not
item.get_name().endswith(".buf") or not item.get_name().endswith(".lg"):
                new_dir_list.append(item)
        for item in self.list:
            if item.get_name().endswith(".bin"):
                new_dir_list.append(item)
        for item in self.list:
            if item.get_name().endswith(".buf"):
                new_dir_list.append(item)
        for item in self.list:
            if item.get_name().endswith(".lg"):
                new_dir_list.append(item)
        self.list = new_dir_list



    def print_list(self):
        #self.sort_list()
        print("~~~~" + self.get_name() + "~~~~")
        for item in self.list:
            if(item.get_name().endswith(".bin")):
                print("\033[31m{}\033[0m".format(item.get_name()))
                continue
            elif(item.get_name().endswith(".buf")):
                print("\033[32m{}\033[0m".format(item.get_name()))
                continue
            elif(item.get_name().endswith(".lg")):
```

```
                print("\033[33m{}\033[0m".format(item.get_name()))
                continue
            else:
                print("\033[34m{}\033[0m".format(item.get_name()))




    # Destruc
    def delete(self):
        if(self.__name == "None"):
            raise FileExistsError("Dir not exists")
        print(self.get_name() + " was removed")
        self.__name = "None"
        self.list = list()
```

LogTextFile.py

```
from directory import Directory


class LogTextFile:
    #Constr
    def __init__(self):
        self.__name = "Logs.lg"
        self.context = "Begginning:"
        self.__directory = None
        self.append_context("\n" + self.get_name() + ": created")


    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name +".lg"
        self.append_context("\n" + self.get_name() + ": was renamed")


    #Move
    def move(self, new_repo):
        new_repo.list.append(self)
        if not self.__directory == None:
            self.__directory.list.remove(self)
        self.__directory = new_repo
        self.append_context("\n" + self.get_name() + ": moved to " +
new_repo.get_name())

    #Read
    def get_context(self):
        return self.context

    def get_direcrory_name(self):
        return self.__directory.get_name()

    #Append
    def append_context(self, message: str):
        self.context += message

    # Destruc
    def delete(self):
        if(self.__name == "None"):
            raise FileExistsError("Logger not exists")
```

```
        self.__directory.list.remove(self)
        self.append_context("\n" + self.get_name() + ": was removed")
        self.__name = "None"
```

main.py

```
from binaryFile import BinaryFile
from directory import Directory
from bufferFile import BufferFile
from logtextfile import LogTextFile

log = LogTextFile()
home_dir= Directory("home")
log.move(home_dir)

dir1 = Directory("dir1")
dir1.move_repository(home_dir)
dir2 = Directory("dir2")
dir2.move_repository(dir1)

bin1 = BinaryFile(dir1, log)
bin2 = BinaryFile(dir1, log)
bin3 = BinaryFile(dir2, log)


buf = BufferFile(3, dir1, log)

home_dir.print_list()
dir1.print_list()
dir2.print_list()

#Editing
bin3.set_name("RenamedBin")
print(bin3.get_context())

#Buff
buf.append_queue("Smth1")
buf.append_queue("Smth2")
buf.append_queue("Smth3")
print("~~~~" + buf.get_name() +"~~~~")
print(buf.get_context())

buf.first_out()
print("~~~~" + buf.get_name() +"~~~~")
print(buf.get_context())

#Dir move
dir3 = Directory("dir3")
dir3.move_repository(dir2)
dir2.print_list()
dir2.move_repository(dir3)
dir3.print_list()
dir2.print_list()

#Ending
print("~~~~" + log.get_name() +"~~~~\n" + log.get_context())
```

Завдання 2

BinaryFile.py

```
from directory import Directory
from logtextfile import LogTextFile


class BinaryFile:
    # Constr
    def __init__(self, directory, logg, name):
        if name == "":
            name = "BinaryFile"
        for item in directory.list:
            if item.get_name() == name + ".bin":
                name += "*"
        self.__name = name + ".bin"
        self.context = "Something is here"
        self.__directory = directory
        directory.list.append(self)
        self.log = logg
        self.log.append_context("\n" + self.get_name() + ": created")

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name + ".bin"
        self.log.append_context("\n" + self.get_name() + ": was renamed")

    # Move
    def move(self, new_repo):
        new_repo.list.append(self)
        self.__directory.list.remove(self)
        for item in new_repo.list:
            if item.get_name() == self.__name:
                name = self.__name[0: self.__name.find('.')] + "`" + ".bin"
                self.__name = name
                break
        self.__directory = new_repo
        self.log.append_context("\n" + self.get_name() + ": moved to " +
new_repo.get_name())

    # Read
    def get_context(self):
        return self.context

    def get_direcrory_name(self):
        return self.__directory.get_name()

    # Destruc
    def delete(self):
        if (self.__name == "None"):
            raise FileExistsError("Binary file not exists")
        self.__directory.list.remove(self)
        self.log.append_context("\n" + self.get_name() + ": was removed")
        self.__name = "None"
```

BufferFile.py

```
from directory import Directory
from logtextfile import LogTextFile
```

```
class BufferFile:
    # Constr
    def __init__(self, size, directory, log, name):
        if name == "":
            name = "Buffer"
        for item in directory.list:
            if item.get_name() == name + ".buf":
                name += "*"
        self.__name = name + ".buf"
        self.__size = size
        self.list = list()
        self.__directory = directory
        directory.list.append(self)
        self.log = log
        self.log.append_context("\n" + self.get_name() + ": created")

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name + ".buf"
        self.log.append_context("\n" + self.get_name() + ": was renamed")

    # Move
    def move(self, new_repo):
        new_repo.list.append(self)
        self.__directory.list.remove(self)
        for item in new_repo.list:
            if item.get_name() == self.__name:
                name = self.__name[0: self.__name.find('.')] + "`" + ".buf"
                self.__name = name
                break
        self.__directory = new_repo
        self.log.append_context("\n" + self.get_name() + ": moved to " +
new_repo.get_name())

    # Read
    def get_context(self):
        return self.list

    def get_direcrory_name(self):
        return self.__directory.get_name()

    # Append
    def append_queue(self, item):
        if len(self.list) == self.__size:
            raise OverflowError("max size reached")
        self.list.append(item)
        self.log.append_context("\n" + self.get_name() + ": append queue")

    def first_out(self):
        self.log.append_context("\n" + self.get_name() + ": poped")
        return self.list.pop(0)

    # Destruc
    def delete(self):
        if (self.__name == "None"):
            raise FileExistsError("Buffer not exist")
        self.__directory.list.remove(self)
        self.log.append_context("\n" + self.get_name() + ": was removed")
        self.__name = "None"
```

Directory.py

```
class Directory:

    # Constr
    def __init__(self, name="autodir"):
        self.__name = name
        self.list = list()

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name

    # Move
    def move_repository(self, new_repo):
        for item in new_repo.list:
            if item.get_name() == self.__name:
                name = self.__name + "`"
                self.__name = name
                break
        if new_repo in self.list:
            self.list.remove(new_repo)
            new_repo.list.append(self)
            return
        new_repo.list.append(self)
        self.directory = new_repo

    # Read

    def sort_list(self):
        new_dir_list = []
        for item in self.list:
            if not item.get_name().endswith(".bin") or not
item.get_name().endswith(
                    ".buf") or not item.get_name().endswith(".lg"):
                new_dir_list.append(item)
        for item in self.list:
            if item.get_name().endswith(".bin"):
                new_dir_list.append(item)
        for item in self.list:
            if item.get_name().endswith(".buf"):
                new_dir_list.append(item)
        for item in self.list:
            if item.get_name().endswith(".lg"):
                new_dir_list.append(item)
        self.list = new_dir_list
        return new_dir_list

    def print_list(self):
        # self.sort_list()
        print("~~~~" + self.get_name() + "~~~~")
        for item in self.list:
            if (item.get_name().endswith(".bin")):
                print("\033[31m{}\033[0m".format(item.get_name()))
                continue
            elif (item.get_name().endswith(".buf")):
                print("\033[32m{}\033[0m".format(item.get_name()))
                continue
            elif (item.get_name().endswith(".lg")):
                print("\033[33m{}\033[0m".format(item.get_name()))
                continue
            else:
                print("\033[34m{}\033[0m".format(item.get_name()))
```

```
    # Destruc
    def delete(self):
        if (self.__name == "None"):
            raise FileExistsError("Dir not exists")
        print(self.get_name() + " was removed")
        self.__name = "None"
        self.list = list()

    def delete_directory(self, dir):
        if dir in self.list:
            self.list.remove(dir)



dir = Directory()
```

LogTextFile.py

```
from directory import Directory


class LogTextFile:
    # Constr
    def __init__(self):
        self.__name = "Logs.lg"
        self.context = "Begginning:"
        self.__directory = None
        self.append_context("\n" + self.get_name() + ": created")

    def get_name(self):
        return self.__name

    def set_name(self, name):
        self.__name = name + ".lg"
        self.append_context("\n" + self.get_name() + ": was renamed")

    # Move
    def move(self, new_repo):
        new_repo.list.append(self)
        if not self.__directory == None:
            self.__directory.list.remove(self)
        self.__directory = new_repo
        self.append_context("\n" + self.get_name() + ": moved to " +
new_repo.get_name())

    # Read
    def get_context(self):
        return self.context

    def get_direcrory_name(self):
        return self.__directory.get_name()

    # Append
    def append_context(self, message: str):
        self.context += message

    # Destruc
    def delete(self):
        if (self.__name == "None"):
            raise FileExistsError("Logger not exists")
        self.__directory.list.remove(self)
```

```
        self.append_context("\n" + self.get_name() + ": was removed")
        self.__name = "None"
```

main.py

```
from flask import Flask, render_template, request, url_for, redirect
from binary_file import BinaryFile
from directory import Directory
from bufferFile import BufferFile
from logtextfile import LogTextFile

app = Flask(__name__)
home = Directory("home")
log = LogTextFile()
log.move(home)
binary = BinaryFile(home, log, "Binn")
binaryf = BinaryFile(home, log, "Bins")
bufferfile = BufferFile(5, home, log, "")
nested_dir = Directory("nested_dir")
nested_dir.move_repository(home)


@app.route('/')
def main_page():
    return render_template("main_page.html", home=home)


#####Bin block
@app.route('/binaryfile')
def binaryfile_page():
    return render_template("binaryfile.html")


@app.route('/binaryfile_create', methods=['POST', 'GET'])
def binaryfile_create():
    if request.method == 'POST':
        name = request.form['name']
        binaryfile = BinaryFile(home, log, name)
        return redirect('/')
    else:
        return render_template('binaryfile_create.html')


@app.route('/binaryfile_read/<string:name>')
def binaryfile_read(name):
    binF = ""
    for item in home.list:
        if item.get_name() == name + ".bin":
            binF = item
    if binF == "":
        return redirect('/binaryfile')
    text = binF.get_context()
    return render_template("binaryfile_read.html", text=text)


@app.route('/binaryfile_del/<string:name>', methods=['POST', 'GET'])
def binaryfile_delete(name):
    binF = ""
    for item in home.list:
        if item.get_name() == name + ".bin":
            binF = item
    if binF == "":
```

```
        return redirect('/binaryfile')

    if request.method == 'POST':
        binF.delete()
        return redirect('/')
    else:
        return render_template("binaryfile_del.html")


@app.route('/binaryfile_move/<string:name>', methods=['POST', 'GET'])
def binaryfile_move(name):
    binF = ""
    for item in home.list:
        if item.get_name() == name + ".bin":
            binF = item
    if binF == "":
        return redirect('/binaryfile')

    if request.method == 'POST':
        name = request.form['name']
        dir = ""
        for item in home.list:
            if item.get_name() == name:
                dir = item
        if dir == "":
            return redirect('/binaryfile')

        binF.move(dir)
        return redirect('/')
    else:
        return render_template("binaryfile_move.html")



#####Buf block
@app.route('/bufferfile')
def bufferfile_page():
    return render_template("bufferfile.html")


@app.route('/bufferfile_create', methods=['POST', 'GET'])
def bufferfile_create():
    if request.method == 'POST':
        name = request.form['name']
        size = request.form['size']
        bufferfile = BufferFile(size, home, log, name)
        return redirect('/')
    else:
        return render_template('bufferfile_create.html')


@app.route('/bufferfile_read/<string:name>')
def bufferfile_read(name):
    bufF = ""
    for item in home.list:
        if item.get_name() == name + ".buf":
            bufF = item
    if bufF == "":
        return redirect('/bufferfile')
    text = bufF.get_context()
    return render_template("bufferfile_read.html", text=text)


@app.route('/bufferfile_add/<string:name_buf>', methods=['POST', 'GET'])
```

```
def bufferfile_add(name_buf):
    bufF = ""
    for item in home.list:
        if item.get_name() == name_buf + ".buf":
            bufF = item
    if bufF == "":
        return redirect('/bufferfile')

    if request.method == 'POST':
        name = request.form['name']
        bufF.append_queue(name)
        return redirect('/bufferfile_read/<string:name>')
    else:
        return render_template('bufferfile_add.html')


@app.route('/bufferfile_pop/<string:name_buf>', methods=['POST', 'GET'])
def bufferfile_pop(name_buf):
    bufF = ""
    for item in home.list:
        if item.get_name() == name_buf + ".buf":
            bufF = item
    if bufF == "":
        return redirect('/bufferfile')

    if request.method == 'POST':
        bufF.first_out()
        return redirect('/bufferfile_read/<string:name>')
    else:
        return render_template('bufferfile_pop.html')


@app.route('/bufferfile_del/<string:name>', methods=['POST', 'GET'])
def bufferfile_delete(name):
    bufF = ""
    for item in home.list:
        if item.get_name() == name + ".buf":
            bufF = item
    if bufF == "":
        return redirect('/bufferfile')

    if request.method == 'POST':
        bufF.delete()
        return redirect('/')
    else:
        return render_template("bufferfile_del.html")


@app.route('/bufferfile_move/<string:name>', methods=['POST', 'GET'])
def bufferfile_move(name):
    bufF = ""
    for item in home.list:
        if item.get_name() == name + ".buf":
            bufF = item
    if bufF == "":
        return redirect('/bufferfile')

    if request.method == 'POST':
        name = request.form['name']
        dir = ""
        for item in home.list:
            if item.get_name() == name:
                dir = item
        if dir == "":
```

```
            return redirect('/bufferfile')

        bufF.move(dir)
        return redirect('/')
    else:
        return render_template("bufferfile_move.html")



#####Log block
@app.route('/logtextfile')
def logtextfile_page():
    return render_template("logtextfile.html")


@app.route('/logtextfile_create', methods=['POST', 'GET'])
def logtextfile_create():
    if request.method == 'POST':
        return redirect('/')
    else:
        return render_template('logtextfile_create.html')


@app.route('/logtextfile_read')
def logtextfile_read():
    text = log.get_context()
    return render_template("logtextfile_read.html", text=text)


@app.route('/logtextfile_del', methods=['POST', 'GET'])
def logtextfile_delete():
    if request.method == 'POST':
        return redirect('/')
    else:
        return render_template("logtextfile_del.html")



### Directory block
@app.route('/directory')
def directory_page():
    return render_template("directory.html")


@app.route('/directory_create', methods=['POST', 'GET'])
def directory_create():
    if request.method == 'POST':
        name = request.form['name']
        directory = Directory(name)
        directory.move_repository(home)
        return redirect('/')
    else:
        return render_template('directory_create.html')


@app.route('/directory_read/<string:name>')
def directory_read(name):
    dirF = ""
    for item in home.list:
        if item.get_name() == name:
            dirF = item
    if dirF == "":
        return redirect('/directory')
    list = dirF.list
```

```
    return render_template("directory_read.html", list=list)


@app.route('/directory_del/<string:name>', methods=['POST', 'GET'])
def directory_delete(name):
    dirF = ""
    for item in home.list:
        if item.get_name() == name:
            dirF = item
    if dirF == "":
        return redirect('/directory')

    if request.method == 'POST':
        home.delete_directory(dirF)
        dirF.delete()
        return redirect('/')
    else:
        return render_template("directory_del.html")


@app.route('/directory_move/<string:name>', methods=['POST', 'GET'])
def directory_move(name):
    dirF = ""
    for item in home.list:
        if item.get_name() == name:
            dirF = item
    if dirF == "":
        return redirect('/directory')

    if request.method == 'POST':
        name = request.form['name']
        dir = ""
        for item in home.list:
            if item.get_name() == name:
                dir = item
        if dir == "":
            return redirect('/directory')
        home.delete_directory(dirF)
        dirF.move_repository(dir)
        return redirect('/')
    else:
        return render_template("directory_move.html")



if __name__ == "__main__":
    app.run(debug=True)
```

Шаблони з використанням Flask доступні за посиланням:

https://github.com/BelitskyiAlexandr/qa-kp01-Belitskyi/tree/main/sem_1_lab_2/templates

test_server.py

```
import requests
import json
from flask import Flask, render_template, request, url_for, redirect
```

```
from main import app

flask_app = app


def test_main_page():
    with flask_app.test_client() as test_client:
        response = test_client.get('/')
        assert response.status_code == 200


def test_binary_pages():
    with flask_app.test_client() as test_client:
        response = test_client.get('/binaryfile')
        assert response.status_code == 200
        response = test_client.get('/binaryfile_create')
        assert response.status_code == 200

        response = test_client.get('/binaryfile_read/<string:name>')
        assert response.status_code == 302
        response = test_client.get('/binaryfile_del/<string:name>')
        assert response.status_code == 302
        response = test_client.get('/binaryfile_move/<string:name>')
        assert response.status_code == 302

        response = test_client.get('/binaryfile_read')
        assert response.status_code == 404
        response = test_client.get('/binaryfile_del')
        assert response.status_code == 404
        response = test_client.get('/binaryfile_move')
        assert response.status_code == 404


def test_buffer_pages():
    with flask_app.test_client() as test_client:
        response = test_client.get('/bufferfile')
        assert response.status_code == 200
        response = test_client.get('/bufferfile_create')
        assert response.status_code == 200

        response = test_client.get('/bufferfile_read/<string:name>')
        assert response.status_code == 302
        response = test_client.get('/bufferfile_del/<string:name>')
        assert response.status_code == 302
        response = test_client.get('/bufferfile_move/<string:name>')
        assert response.status_code == 302
        response = test_client.get('/bufferfile_add/<string:name>')
        assert response.status_code == 302
        response = test_client.get('/bufferfile_pop/<string:name>')
        assert response.status_code == 302

        response = test_client.get('/bufferfile_read')
        assert response.status_code == 404
        response = test_client.get('/bufferfile_del')
        assert response.status_code == 404
        response = test_client.get('/bufferfile_move')
        assert response.status_code == 404
        response = test_client.get('/bufferfile_add')
        assert response.status_code == 404
        response = test_client.get('/bufferfile_pop')
        assert response.status_code == 404


def test_logtextfile_pages():
```

```
    with flask_app.test_client() as test_client:
        response = test_client.get('/logtextfile')
        assert response.status_code == 200
        response = test_client.get('/logtextfile_create')
        assert response.status_code == 200
        response = test_client.get('/logtextfile_del')
        assert response.status_code == 200
        response = test_client.get('/logtextfile_read')
        assert response.status_code == 200

        response = test_client.get('/logtextfile_del/<string:name>')
        assert response.status_code == 404
        response = test_client.get('/logtextfile_read/<string:name>')
        assert response.status_code == 404
        response = test_client.get('/logtextfile_move')
        assert response.status_code == 404


def test_directory_pages():
    with flask_app.test_client() as test_client:
        response = test_client.get('/directory')
        assert response.status_code == 200
        response = test_client.get('/directory_create')
        assert response.status_code == 200

        response = test_client.get('/directory_read/<string:name>')
        assert response.status_code == 302
        response = test_client.get('/directory_del/<string:name>')
        assert response.status_code == 302
        response = test_client.get('/directory_move/<string:name>')
        assert response.status_code == 302

        response = test_client.get('/directory_read')
        assert response.status_code == 404
        response = test_client.get('/directory_del')
        assert response.status_code == 404
        response = test_client.get('/directory_move')
        assert response.status_code == 404


def test_binaryfile_create():
    ENDPOINT = "http://127.0.0.1:5000/binaryfile_create"
    response = requests.get(ENDPOINT)

    #data = response.text
    #print(data)

    payload = {
        "name": "Binn"
    }
    response = requests.post(ENDPOINT, json=payload)
    data = response.text
    print(data)

    name = "Binn"
    check_existance_response =
requests.get(f"http://127.0.0.1:5000/binaryfile_read/{name}")
    assert check_existance_response.status_code == 200
    data = check_existance_response.text
    print(data)


def test_binary_delete():
    name = "Binary"
```

```
    ENDPOINT = f"http://127.0.0.1:5000/binaryfile_del/{name}"
    response = requests.post(ENDPOINT)

    check_existance_response =
requests.get(f"http://127.0.0.1:5000/binaryfile_read/{name}")
    assert check_existance_response.status_code == 200      #redirect
    data = check_existance_response.text
    print(data)


def test_binary_move():
    name = "Binary"
    ENDPOINT = f"http://127.0.0.1:5000/binaryfile_move/{name}"
    payload = {
        "name": "home"
    }
    response = requests.post(ENDPOINT, json=payload)

    check_existance_response =
requests.get(f"http://127.0.0.1:5000/binaryfile_read/{name}")
    assert check_existance_response.status_code == 200  # redirect
    data = check_existance_response.text
    print(data)



def test_bufferfile_create():
    ENDPOINT = "http://127.0.0.1:5000/bufferfile_create"
    response = requests.get(ENDPOINT)

    payload = {
        "name": "Buffer"
    }
    response = requests.post(ENDPOINT, json=payload)
    data = response.text
    print(data)

    name = "Buffer"
    check_existance_response =
requests.get(f"http://127.0.0.1:5000/bufferfile_read/{name}")
    assert check_existance_response.status_code == 200
    data = check_existance_response.text
    print(data)


def test_buffer_add():
    name = "Buffer"
    ENDPOINT = f"http://127.0.0.1:5000/bufferfile_add/{name}"
    payload = {
        "name": "smth"
    }
    response = requests.post(ENDPOINT, json=payload)

    check_existance_response =
requests.get(f"http://127.0.0.1:5000/bufferfile_read/{name}")
    assert check_existance_response.status_code == 200  # redirect
    data = check_existance_response.text
    print(data)


def test_buffer_pop():
    name = "Buffer"
    ENDPOINT = f"http://127.0.0.1:5000/bufferfile_pop/{name}"
    response = requests.post(ENDPOINT)
```

```
    check_existance_response =
requests.get(f"http://127.0.0.1:5000/bufferfile_read/{name}")
    assert check_existance_response.status_code == 200  # redirect
    data = check_existance_response.text
    print(data)

def test_bufferfile_delete():
    name = "Buffer"
    ENDPOINT = f"http://127.0.0.1:5000/bufferfile_del/{name}"
    response = requests.post(ENDPOINT)

    check_existance_response =
requests.get(f"http://127.0.0.1:5000/bufferfile_read/{name}")
    assert check_existance_response.status_code == 200      #redirect
    data = check_existance_response.text
    print(data)


def test_buffer_move():
    name = "Buffer"
    ENDPOINT = f"http://127.0.0.1:5000/bufferfile_move/{name}"
    payload = {
        "name": "home"
    }
    response = requests.post(ENDPOINT, json=payload)

    check_existance_response =
requests.get(f"http://127.0.0.1:5000/bufferfile_read/{name}")
    assert check_existance_response.status_code == 200  # redirect
    data = check_existance_response.text
    print(data)


def test_logtextfile_create():
    ENDPOINT = "http://127.0.0.1:5000/logtextfile_create"
    response = requests.get(ENDPOINT)
    assert response.status_code == 200
    data = response.text
    print(data)


def test_logtextfile_delete():
    ENDPOINT = "http://127.0.0.1:5000/logtextfile_del"
    response = requests.get(ENDPOINT)

    assert response.status_code == 200
    data = response.text
    print(data)


def test_logtextfile_move():
    ENDPOINT = "http://127.0.0.1:5000/bufferfile_move"
    response = requests.get(ENDPOINT)
    assert response.status_code == 404


def test_binaryfile_create():
    ENDPOINT = "http://127.0.0.1:5000/directory_create"
    response = requests.get(ENDPOINT)

    payload = {
        "name": "Directory"
    }
```

```
    response = requests.post(ENDPOINT, json=payload)
    data = response.text
    print(data)

    name = "Directory"
    check_existance_response =
requests.get(f"http://127.0.0.1:5000/directory_read/{name}")
    assert check_existance_response.status_code == 200
    data = check_existance_response.text
    print(data)


def test_directory_delete():
    name = "Directory"
    ENDPOINT = f"http://127.0.0.1:5000/directory_del/{name}"
    response = requests.post(ENDPOINT)

    check_existance_response =
requests.get(f"http://127.0.0.1:5000/directory_read/{name}")
    assert check_existance_response.status_code == 200      #redirect
    data = check_existance_response.text
    print(data)


def test_directory_move():
    name = "Directory"
    ENDPOINT = f"http://127.0.0.1:5000/directory_move/{name}"
    payload = {
        "name": "dir1"
    }
    response = requests.post(ENDPOINT, json=payload)

    check_existance_response =
requests.get(f"http://127.0.0.1:5000/directory_read/{name}")
    assert check_existance_response.status_code == 200  # redirect
    data = check_existance_response.text
    print(data)
```

test_bufferfile.py

```
import pytest

from bufferFile import BufferFile
from directory import Directory
from logtextfile import LogTextFile

def test_init():
    with pytest.raises(TypeError):
        BufferFile()


def test_name():
    dir = Directory()
    log = LogTextFile()
    buf = BufferFile(5, dir, log, "")
    assert buf.get_name() == "Buffer.buf"
    buf.set_name("New name")
    assert buf.get_name() == "New name.buf"


def test_good_move():
```

```
    dir1 = Directory("dir1")
    dir2 = Directory("dir2")
    log = LogTextFile()
    buf = BufferFile(5, dir1, log, "")
    buf.move(dir2)
    assert buf.get_direcrory_name() == "dir2"

def test_name_after_move():
    dir1 = Directory("dir1")
    dir2 = Directory("dir2")
    log = LogTextFile()
    buf1 = BufferFile(5, dir1, log, "")
    buf2 = BufferFile(5, dir2, log, "")
    buf1.move(dir2)
    assert buf1.get_name() == "Buffer`.buf"

def test_content():
    dir = Directory()
    log = LogTextFile()
    buf = BufferFile(5, dir, log, "")
    assert buf.get_context() == []


def test_delete():
    dir = Directory()
    log = LogTextFile()
    buf = BufferFile(5, dir, log, "")
    buf.delete()
    assert dir.list == []


def test_queue():
    dir = Directory()
    log = LogTextFile()
    buf = BufferFile(1, dir, log, "")
    buf.append_queue("ss")
    with pytest.raises(OverflowError):
        buf.append_queue("qq")


def test_pop():
    dir = Directory()
    log = LogTextFile()
    buf = BufferFile(1, dir, log, "")
    buf.append_queue("ss")
    assert buf.first_out() == "ss"
    assert buf.list == []


def test_redelete():
    dir = Directory()
    log = LogTextFile()
    buf = BufferFile(3, dir, log, "")
    buf.delete()
    with pytest.raises(FileExistsError):
        buf.delete()
```

test_binaryFile.py

```
import pytest

from binary_file import BinaryFile
from directory import Directory
from logtextfile import LogTextFile

def test_init():
    with pytest.raises(TypeError):
        BinaryFile()

def test_name():
    dir = Directory()
    log = LogTextFile()
    bin = BinaryFile(dir, log, "")
    assert bin.get_name() == "BinaryFile.bin"
    bin.set_name("New name")
    assert bin.get_name() == "New name.bin"


def test_good_move():
    dir1 = Directory("dir1")
    dir2 = Directory("dir2")
    log = LogTextFile()
    bin = BinaryFile(dir1, log, "")
    bin.move(dir2)
    assert bin.get_direcrory_name() == "dir2"


def test_name_after_move():
    dir1 = Directory("dir1")
    dir2 = Directory("dir2")
    log = LogTextFile()
    bin1 = BinaryFile(dir1, log, "")
    bin2 = BinaryFile(dir2, log, "")
    bin1.move(dir2)
    assert bin1.get_name() == "BinaryFile`.bin"


def test_content():
    dir = Directory()
    log = LogTextFile()
    bin = BinaryFile(dir, log, "")
    assert bin.get_context() == "Something is here"


def test_delete():
    dir = Directory()
    log = LogTextFile()
    bin = BinaryFile(dir, log, "")
    bin.delete()
    assert dir.list == []


def test_redelete():
    dir = Directory()
    log = LogTextFile()
    bin = BinaryFile(dir, log, "")
    bin.delete()
    with pytest.raises(FileExistsError):
        bin.delete()
```

test_directory.py

```
import pytest

from directory import Directory


def test_init():
    dir = Directory()
    assert dir.get_name() == "autodir"
    dir.set_name("dir1")
    assert dir.get_name() == "dir1"


def test_good_move():
    dir1 = Directory("dir1")
    dir2 = Directory("dir2")
    dir1.move_repository(dir2)
    assert dir1 in dir2.list


def test_name_after_move():
    dir1 = Directory("dir1")
    dir2 = Directory("dir1")
    dir3 = Directory("dir1")
    dir1.move_repository(dir3)
    dir2.move_repository(dir3)
    assert dir2.get_name() == "dir1`"


def test_delete():
    dir = Directory()
    dir.delete()
    assert dir.list == []
    assert dir.get_name() == "None"


def test_redelete():
    dir = Directory()
    dir.delete()
    with pytest.raises(FileExistsError):
        dir.delete()
```

test_logtextfile.py

```
import pytest

from directory import Directory
from logtextfile import LogTextFile



def test_name():
    log = LogTextFile()
    assert log.get_name() == "Logs.lg"
    log.set_name("New name")
    assert log.get_name() == "New name.lg"


def test_good_move():
    dir1 = Directory("dir1")
    dir2 = Directory("dir2")
    log = LogTextFile()
```

```
    log.move(dir2)
    assert log.get_direcrory_name() == "dir2"


def test_content():
    log = LogTextFile()
    assert log.get_context() == "Begginning:\nLogs.lg: created"
    log.append_context(" + some")
    assert log.get_context() == "Begginning:\nLogs.lg: created + some"


def test_delete():
    dir = Directory()
    log = LogTextFile()
    log.move(dir)
    log.delete()
    assert dir.list == []


def test_redelete():
    dir = Directory()
    log = LogTextFile()
    log.move(dir)
    log.delete()
    with pytest.raises(FileExistsError):
        log.delete()
```

Завдання 3

main.py

```
from binary_file import BinaryFile
from directory import Directory
from bufferFile import BufferFile
from logtextfile import LogTextFile

home = Directory("home")
log = LogTextFile()
log.move(home)
binary = BinaryFile(home, log, "Binn")
binaryf = BinaryFile(home, log, "Bins")
bufferfile = BufferFile(5, home, log, "")
nested_dir = Directory("nested_dir")
nested_dir.move_repository(home)
try:
    while True:
        command = input('Enter command: ')
        command = command.split()
        if command[0] == 'exit':
            print("Exiting...")
            break

        # directory
        elif command[0] == 'mkdir':  # make
            if len(command) != 2:
                print("Incorrect number of args " + str(len(command)) + ",
but need: mkdir {name}")
                continue
            dir = Directory(command[1])
            dir.move_repository(home)
            print("okmkdir")
```

```
        elif command[0] == 'mvdir':  # move
            if len(command) != 3:
                print(
                    "Incorrect number of args " + str(len(command)) + ", but
need: mvdir {name_of_moving} {name_of_moveIn}")
                continue

            dirF = ""
            for item in home.list:
                if item.get_name() == command[1]:
                    dirF = item
            if dirF == "":
                print("Directory " + str(command[1]) + " does not exist")
                continue

            dir = ""
            for item in home.list:
                if item.get_name() == command[2]:
                    dir = item
            if dir == "":
                print("Directory " + str(command[2]) + " does not exist")
                continue

            home.delete_directory(dirF)
            dirF.move_repository(dir)

            print("okmvdir")

        elif command[0] == 'rddir':  # read
            if len(command) != 2:
                print("Incorrect number of args " + str(len(command)) + ",
but need: rddir {name}")
                continue

            if command[1] == "home":
                home.print_list()
                continue
            dirF = ""
            for item in home.list:
                if item.get_name() == command[1]:
                    dirF = item

            if dirF == "":
                print("Directory " + str(command[1]) + " does not exist")
                continue
            dirF.print_list()
            print("okrddir")

        elif command[0] == 'deldir':  # delete
            if len(command) != 2:
                print("Incorrect number of args " + str(len(command)) + ",
but need: deldir {name}")
                continue
            dirF = ""
            for item in home.list:
                if item.get_name() == command[1]:
                    dirF = item
            if dirF == "":
                print("Directory " + str(command[1]) + " does not exist")
                continue

            home.delete_directory(dirF)
            dirF.delete()
            print("okdeldir")
```

```
        # binary
        elif command[0] == 'mkbin':  # make
            if len(command) != 2:
                print("Incorrect number of args " + str(len(command)) + ",
but need: mkbin {name}")
                continue
            binaryfile = BinaryFile(home, log, command[1])
            print("okmkbin")

        elif command[0] == 'mvbin':  # move
            if len(command) != 3:
                print(
                    "Incorrect number of args " + str(len(command)) + ", but
need: mvbin {name_of_moving} {name_of_moveIn}")
                continue

            binF = ""
            for item in home.list:
                if item.get_name() == command[1] + ".bin":
                    binF = item
            if binF == "":
                print("BinaryFile " + str(command[1]) + " does not exist")
                continue

            dir = ""
            for item in home.list:
                if item.get_name() == command[2]:
                    dir = item
            if dir == "":
                print("Directory " + str(command[2]) + " does not exist")
                continue

            binF.move(dir)

            print("okmvbin")

        elif command[0] == 'rdbin':  # read
            if len(command) != 2:
                print(
                    "Incorrect number of args " + str(len(command)) + ", but
need: rdbin {name}")
                continue

            binF = ""
            for item in home.list:
                if item.get_name() == command[1] + ".bin":
                    binF = item
            if binF == "":
                print("BinaryFile " + str(command[1]) + " does not exist")
                continue

            print(binF.get_context())
            print("okrdbin")

        elif command[0] == 'delbin':  # delete
            if len(command) != 2:
                print(
                    "Incorrect number of args " + str(len(command)) + ", but
need: delbin {name}")
                continue

            binF = ""
```

```
            for item in home.list:
                if item.get_name() == command[1] + ".bin":
                    binF = item
            if binF == "":
                print("BinaryFile " + str(command[1]) + " does not exist")
                continue

            binF.delete()

            print("okdelbin")



        # buff
        elif command[0] == 'mkbuf':  # make
            if len(command) != 3:
                print("Incorrect number of args " + str(len(command)) + ",
but need: mkbuf {name} {size}")
                continue
            if not command[2].isdigit():
                print("Size must be number: mkbuf {name} {size}")
                continue
            bufferfile = BufferFile(command[2], home, log, command[1])

            print("okmkbuf")

        elif command[0] == 'appqu':  # add item
            if len(command) != 3:
                print("Incorrect number of args " + str(len(command)) + ",
but need: appqu {name} {item}")
                continue

            bufF = ""
            for item in home.list:
                if item.get_name() == command[1] + ".buf":
                    bufF = item
            if bufF == "":
                print("Buffer " + str(command[1]) + " does not exist")
                continue

            bufF.append_queue(command[2])

            print("okappqubuf")

        elif command[0] == 'popqu':  # pop item
            if len(command) != 2:
                print("Incorrect number of args " + str(len(command)) + ",
but need: popqu {name}")
                continue

            bufF = ""
            for item in home.list:
                if item.get_name() == command[1] + ".buf":
                    bufF = item
            if bufF == "":
                print("Buffer " + str(command[1]) + " does not exist")
                continue

            bufF.first_out()

            print("okpopbuf")

        elif command[0] == 'mvbuf':  # move
            if len(command) != 3:
```

```
                print(
                    "Incorrect number of args " + str(len(command)) + ", but
need: mvbuf {name_of_moving} {name_of_moveIn}")
                continue

            bufF = ""
            for item in home.list:
                if item.get_name() == command[1] + ".buf":
                    bufF = item
            if bufF == "":
                print("Buffer " + str(command[1]) + " does not exist")
                continue

            dir = ""
            for item in home.list:
                if item.get_name() == command[2]:
                    dir = item
            if dir == "":
                print("Directory " + str(command[2]) + " does not exist")
                continue

            bufF.move(dir)

            print("okmvbuf")

        elif command[0] == 'rdbuf':  # read
            if len(command) != 2:
                print("Incorrect number of args " + str(len(command)) + ",
but need: rdbuf {name}")
                continue

            bufF = ""
            for item in home.list:
                if item.get_name() == command[1] + ".buf":
                    bufF = item

            if bufF == "":
                print("BufferFile " + str(command[1]) + " does not exist")
                continue

            for i in range(0, len(bufF.get_context())):
                print(bufF.get_context()[i])

            print("okrdbuf")

        elif command[0] == 'delbuf':  # delete
            if len(command) != 2:
                print("Incorrect number of args " + str(len(command)) + ",
but need: delbuf {name}")
                continue

            bufF = ""
            for item in home.list:
                if item.get_name() == command[1] + ".buf":
                    bufF = item

            if bufF == "":
                print("BufferFile " + str(command[1]) + " does not exist")
                continue

            bufF.delete()

            print("okdelbuf")
```

```
        # log
        elif command[0] == 'mklog':  # make
            if len(command) != 2:
                print("Incorrect number of args " + str(len(command)) + ",
but need: mklog {name}")
                continue

            print("Can be only 1 logger")

        elif command[0] == 'mvlog':  # move
            if len(command) != 2:
                print("Incorrect number of args " + str(len(command)) + ",
but need: mvlog {name}")
                continue

            print("Cannot move logger")

        elif command[0] == 'rdlog':  # read
            if len(command) != 1:
                print("Incorrect number of args " + str(len(command)) + ",
but need: rdlog")
                continue

            for i in range(0, len(log.get_context())):
                print(log.get_context()[i])

        elif command[0] == 'dellog':  # delete
            if len(command) != 1:
                print("Incorrect number of args " + str(len(command)) + ",
but need: dellpg")
                continue

            print("Cannot delete logger")

        # ex
        else:
            print("Unknown command")

except EOFError as er:
    print(er)
```

tests.robot

```
*** Settings ***
Library    Process
Suite Teardown  Terminate All Processes     kill=True

*** Test Cases ***
RunTest
    ${result}=  Start Process  ${CURDIR}/main.py  print('exit')
    Log    ${result.stdout}
    Terminate All Processes

MkDirTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('mkdir')
    Log    ${result.stdout}
    Terminate All Processes

MkDirTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('mkdir gg')
    Log    ${result.stdout}
```

```
    Terminate All Processes

PrintDirTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('rddir home')
    Log    ${result.stdout}
    Terminate All Processes

RdDirTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('rddir')
    Log    ${result.stdout}
    Terminate All Processes

MvDirTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('mvdir gg hh')
    Log    ${result.stdout}
    Terminate All Processes

MvDirTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('mvdir')
    Log    ${result.stdout}
    ${result}=  Start Process  ${CURDIR}/main.py  print('mvdir gg')
    Log    ${result.stdout}
    Terminate All Processes

DelDirTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('deldir gg')
    Log    ${result.stdout}
    Terminate All Processes

DelDirTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('deldir')
    Log    ${result.stdout}
    Terminate All Processes


#bin
MkBinTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('mkbin')
    Log    ${result.stdout}
    Terminate All Processes

MkBinTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('mkbin gg')
    Log    ${result.stdout}
    Terminate All Processes

PrintBinTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('rdbin home')
    Log    ${result.stdout}
    Terminate All Processes

RdBinTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('rdbin')
    Log    ${result.stdout}
    Terminate All Processes

MvBinTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('mvbin Binn hh')
    Log    ${result.stdout}
    Terminate All Processes

MvBinTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('mvbin')
    Log    ${result.stdout}
```

```
    ${result}=  Start Process  ${CURDIR}/main.py  print('mvbin Binn')
    Log    ${result.stdout}
    Terminate All Processes

DelBinTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('delbin Binn')
    Log    ${result.stdout}
    Terminate All Processes

DelBinTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('delbin')
    Log    ${result.stdout}
    Terminate All Processes


#buf
MkBufTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('mkbuf')
    Log    ${result.stdout}
    Terminate All Processes

MkBufTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('mkbuf Buffer')
    Log    ${result.stdout}
    Terminate All Processes

PrintBufTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('rdbuf Buffer')
    Log    ${result.stdout}
    Terminate All Processes

RdBufTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('rdbuf')
    Log    ${result.stdout}
    Terminate All Processes

MvBufTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('mvbuf Buffer hh')
    Log    ${result.stdout}
    Terminate All Processes

MvBufTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('mvbuf')
    Log    ${result.stdout}
    ${result}=  Start Process  ${CURDIR}/main.py  print('mvbuf Buffer')
    Log    ${result.stdout}
    Terminate All Processes

AppBufTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('appqu Buffer 2')
    Log    ${result.stdout}
    Terminate All Processes

AppBufTestIncorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('appqu Buffer')
    Log    ${result.stdout}
    Terminate All Processes

PopBufTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('popqu Buffer')
    Log    ${result.stdout}
    Terminate All Processes

DelBufTestCorrect
```

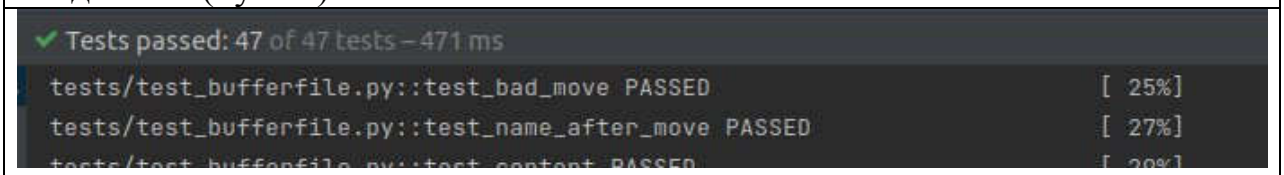
```
    ${result}=  Start Process  ${CURDIR}/main.py  print('delbuf Buffer')
    Log    ${result.stdout}
    Terminate All Processes

DelBufTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('delbuf')
    Log    ${result.stdout}
    Terminate All Processes


#logger
MkLogTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('mklog')
    Log    ${result.stdout}
    Terminate All Processes

MkLogTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('mklog logg')
    Log    ${result.stdout}
    Terminate All Processes

PrintLogTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('rdlog logg')
    Log    ${result.stdout}
    Terminate All Processes

RdLogTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('rdlog')
    Log    ${result.stdout}
    Terminate All Processes

MvLogTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('mvlog Logg hh')
    Log    ${result.stdout}
    Terminate All Processes

MvLogTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('mvlog')
    Log    ${result.stdout}
    ${result}=  Start Process  ${CURDIR}/main.py  print('mvlog Logg')
    Log    ${result.stdout}
    Terminate All Processes

DelLogTestCorrect
    ${result}=  Start Process  ${CURDIR}/main.py  print('dellog Logg')
    Log    ${result.stdout}
    Terminate All Processes

DelLogTestIncorrectValue
    ${result}=  Start Process  ${CURDIR}/main.py  print('dellog')
    Log    ${result.stdout}
    Terminate All Processes

IncorrectCommand
    ${result}=  Start Process  ${CURDIR}/main.py  print('Something here')
    Log    ${result.stdout}
    Terminate All Processes
```

Dockerfile

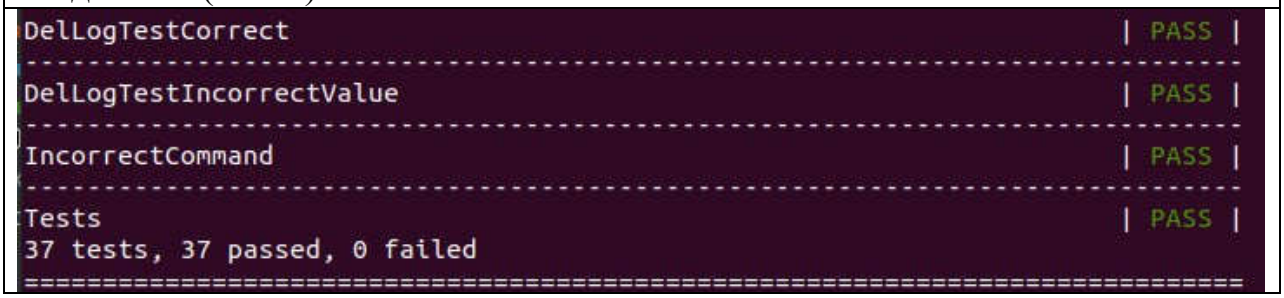
```
FROM python:3

COPY . /sem_1_lab_3

WORKDIR /sem_1_lab_3

RUN python main.py

CMD [ "python3", "main.py" ]
```

Результати виконання тестів

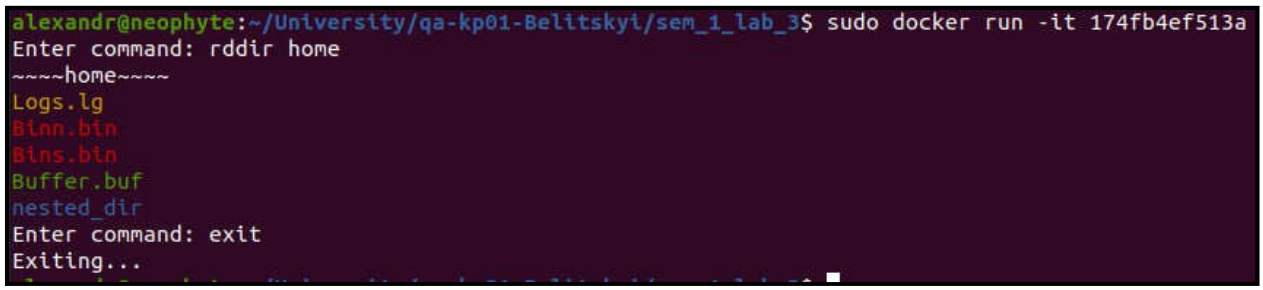
Завдання 2 (PyTest)



Завдання 3 (Robot)



Запуск проєкту через Docker Container



Висновки

В даній курсовій роботі було набуто навички тестування програмного забезпечення, вивчено інструменти для тестування ПЗ (Robot, Pytest), проведено ознайомлення з системою віртуалізації Docker та покращено навички володіння мовою Python. Розроблення файлової системи в пам'яті допомогло зрозуміти архітектуру та принцип роботи файлових систем, що допоможе при роботі з архітектурами файлових систем.

Виконання роботи відбувалося в операційній системі Ubuntu, що побудована на основі Linux. Це допомогло набути навичок володіння терміналом та підвищити швидкість роботи з ним.